Selective Training: A Strategy for Fast Backpropagation on Sentence Embeddings

Neerbek, Jan; Dolog, Peter; Assent, Ira

Published in:
Advances in Knowledge Discovery and Data Mining - 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17, 2019, Proceedings, Part III

DOI (link to publication from Publisher):
10.1007/978-3-030-16142-2_4

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

Selective Training: A Strategy for Fast Backpropagation on Sentence Embeddings

Jan Neerbek[1,2], Peter Dolog[3], and Ira Assent[1]

[1] Dept. Computer Science, DIGIT center, Aarhus University, Denmark
{jan.neerbek, ira}@cs.au.dk
[2] Alexandra Institute, Denmark
[3] Department of Computer Science, Aalborg University, Denmark
dolog@cs.aau.dk

Abstract. Representation or embedding based machine learning models, such as language models or convolutional neural networks have shown great potential for improved performance. However, for complex models on large datasets training time can be extensive, approaching weeks, which is often infeasible in practice. In this work, we present a method to reduce training time substantially by selecting training instances that provide relevant information for training. Selection is based on the similarity of the learned representations over input instances, thus allowing for learning a non-trivial weighting scheme from multi-dimensional representations. We demonstrate the efficiency and effectivity of our approach in several text classification tasks using recursive neural networks. Our experiments show that by removing approximately one fifth of the training data the objective function converges up to six times faster without sacrificing accuracy.

Keywords: Selective Training, Machine Learning, Neural Network, Recursive Models

1 Introduction

Recent years have seen substantial performance improvements in machine learning for *deep models* in a variety of application domains [10]. However, training times for deep models can easily be in the order of days [27] or even weeks [3]. Being able to efficiently train and evaluate new models is important in order to preserve our ability to investigate and develop better machine learning models. Thus, training effort may be a critical factor in the deployment and advancement of more powerful, expressive machine learning models. This is certainly true for deep neural network models where the quest for stronger and better neural models drives doubling of models sizes (number of neurons) approximately every 2.4 years [10].

In this work, we present *Selective Training*, an effective training strategy for artificial neural network models. In a nutshell, by focusing on instances with relevant information for training, our approach requires fewer training iterations to converge to a stable and effective model. Selective Training adjusts training based on multi-dimensional representations of what the network has learned. It can be used with different training methodologies such as standard backpropagation or adaptive training approaches like Adam where the learning rate is adjusted depending on the loss gradient [14].

In this paper we focus on the deep structured gradient backpropagation training approach, *backpropagation-through-structure (BPTS)* [12, 9] for text classification tasks with high training times, where we observe substantial improvements in training time. Still, our results are not limited to this application, but generalize to classifier models which generate distributed instance representations. We demonstrate in our empirical study on several document datasets that the gains in training time do not come at the cost of accuracy, but may even bring a slight improved accuracy score[4].

Our method identifies obsolete training samples through clustering in representation space. The cluster approach makes it possible to select those parts of the training data that matter for training, and to focus on these in order to reduce training time. This is in contrast to approaches such as instance selection [25] or active learning [28] where the goal is to find the minimum representative instances (or equivalently instance lookups). This difference in goals leads to two major methodological differences; 1) We remove entire clusters rather than instances and 2) we work on embedded representations rather than on instance features. A major challenge for instance selection as reported in [25] is the need for comparing new instances to all previously selected instances. This costly comparison is is a challenge for scale-up, that our method does not suffer from. Clustering into relatively few clusters is sufficient and efficient, and can even be further scaled up using sub-sampling or hierarchical approaches [2].

Our contributions include a selection strategy for training with substantial speed up while still maintaining high model accuracy, an empirical study on four real world datasets that demonstrates the effectiveness and efficiency of our approach, robustness with respect to parameterization, as well as a detailed error analysis.

2 Background

In the following, we study efficient training for complex deep models for text classification, as a concrete instance of costly training problems in deep learning. In the text classification, distributed word embeddings [1] have been immensely successful for a wide range of tasks including sentiment analysis [31], POS-tagging [4] and text classification [13]. Many approaches learn unsupervised word embeddings on large document sets such as word2vec [23] and GloVe [26]. VecAvg proposed to define sentence embeddings as the average of all word embeddings in a given sentence [20]. VecAvg has since been superseded by more expressive models such as recurrent neural networks (RNN) using gated memory cells such as the LSTM [11].

A generalization of the recurrent model has been proposed in [30] as *recursive neural networks (RecNN)*[5]. RecNN models can incorporate semantic knowledge about the sentence in a tree or graph like structure. To evaluate a RecNN a walk is required from node to node through the entire tree. This may negatively impact performance, and the walk may be hard to parallelize, therefore various restricted versions of the RecNN have been proposed such as Hierarchical ConvNet [5] and Graph Convolutional Networks (GCN) [15] where the number of steps in the graph is restricted to a fixed con-

[4] code `https://bitbucket.alexandra.dk/projects/TAB`, data `https://dataverse.harvard.edu/dataverse/enron-w-trees`

[5] In this work we refer to recursive neural networks as *RecNN* to avoid name clash with RNNs.

stant number. In our experimental study, we consider embeddings generated by the full complexity of the RecNN model over the constituency parse trees as proposed in [29]. As we only assume embeddings, our approach can generally be applied to any text or sentence embedding generating approach as well.

3 Problem Definition

Given a dataset D of n text documents $D = \{d_1, d_2, \ldots, d_n\}$, a ground-truth labeling $L : D \rightarrow \{1, 2, \ldots C\}$ with C classes, and label $L(x)$ for $x \in D$, the goal is to train a given model m using as few training cycles as possible while maintaining m's accuracy. Model m is parametrized by a set of parameters $\theta \in \Theta$, where Θ denotes all possible model parametrizations. The approach used to find a good model is referred to as the learning approach, which we denote T_m. For example, T_m could be the application of backpropagation on model m. Given (mini-)batches of b texts per cycle the learning approach updates the set of parameters $T_m : (\Theta, D^b) \rightarrow \Theta$, i.e., given a parametrized model m_θ and a subset $D' \subseteq D$ of b texts the training function returns a new set of parameters θ': $\theta' = T_m(\theta, D')$.

Efficiency of the training approach is then the expected number of random batches of training text documents that needs to be processed by T_m before the performance of the model m_θ converges. That is, further batches do not bring m_θ closer to L, as indicated by an error measure, such as the squared error function $(L(x) - m_\theta(x))^2$.

We wish to minimize the objective function $obj(m_\theta) = \frac{1}{|D|} \sum_{x \in D} (L(x) - m_\theta(x))^2$. Given a threshold ϵ, model $m_{\theta'}$ as the model after training m_θ on additional δ batches, then we say that the model m has converged iff $obj(m_\theta) - obj(m_{\theta'}) < \epsilon$, i.e., further batches do not improve m_θ. In this paper, we use the expected number of batches to produce a converged model on the training dataset as a measure of the training time, denoted $t(T_m)$ or just t_m for short. We define the training time optimization problem for a dataset D with labeling function L, a model architecture m and training method T_m as the minimization of $t(T_m)$ under the constraint that model accuracy be preserved.

For embedding based approaches, a distributed representation of the input is learned in order to predict the correct label. Thus, our model m_θ can be split into a predictive part m^p_θ and an embedding generating part m^e_θ as follows: $m_\theta = m^p_\theta m^e_\theta$. In complex models, the embedding layer in fact typically consists of several layers $m^e_\theta = m^L_\theta \ldots m^2_\theta m^1_\theta$. For our method, we only use the most informative embedding which is the final embedding produced just before a prediction is made. Evaluating model m_θ on training instance x yields $m_\theta(x) = m^p_\theta m^e_\theta(x)$. We refer to the output of the embedding layer as a *representation* of x: $repr(x) = m^e_\theta(x)$.

The key idea in this work is to exploit this representation as a means to identify a subset of the training instances that does not contain information for training. Excluding it from further training reduces training time $t(T_m)$ while not hurting accuracy.

4 Our Approach

To minimize the number of training cycles $t(T_m)$, our goal is to select the most informative training samples. Given a model m_θ and input x, we extract representations

$repr(x)$ of what the model has learned about the input x. In a deep neural network, this generally is the second last layer before the final prediction layer.

If the model has learned to differentiate between two inputs x and x' with different labels $L(x) \neq L(x')$, then their representations should differ as well. If the representations were similar then it would be challenging for the model prediction layer to distinguish between the two representations. We observe that the structure of the representation space allows us to select training instances of interest, namely groups of instances with different labels in close vicinity in representation space, as those have not yet been properly differentiated with respect to their labels. Clearly, training on instances with similar representations and identical label provides less information for learning to distinguish between classes.

We therefore pose the problem of finding subsets of interesting training samples as a clustering problem. While any hard clustering method could be used, we use the well established K-means [21, 7, 2] as a simple and efficient choice. In brief, K-means assigns representations into k clusters such that the total sum (TS) of $L2$ distances between cluster centroid and cluster members is minimized: $TS = \sum_{x_i \in D} \|repr(x_i) - c_j(x_i)\|^2$, where $c_j(x_i)$ is the centroid of the cluster that input x_i is assigned to.

Let $C = \{C_1, C_2, \ldots, C_k\}$ be the resulting clusters, grouped exclusively based on the similarity in representation space. We now analyze them with respect to label purity to identify sets of samples in representation space that have already been learned and can be omitted from further training. We formalize this analysis as the ratio of the *most frequently occurring (MFO)* class label ratio in a cluster, MFO_i, as $MFO_i = \max_{\ell \in L} \frac{\sum_{x \in C_i : L(x) = \ell} 1}{|C_i|}$ i.e., the ratio of the most frequently occurring class label in a cluster C_i is the maximum (over possible labels) of the ratio between the number of instances with that label and the cardinality of the cluster. Clusters with low MFO_i are valuable for training, whereas those where MFO_i is close to 1 are uninteresting as little more can be learned. A strong model has MFO_i for all clusters close to 1 (otherwise accuracy is low, see above). If there are only two classes $\{0, 1\}$, we simplify using the ratio for only one class $f_i = \frac{\sum_{x \in C_i : L(x) = 1} 1}{|C_i|}$ and obtain $MFO_i = \max(f_i, 1 - f_i)$.

Our goal is to separate interesting from uninteresting clusters by finding a suitable threshold for the MFO ratio to filter uninteresting training instances away and focus training on interesting ones only. More formally we wish to choose the lowest possible MFO threshold such that models trained with our approach m^s_θ satisfy the optimization goal, i.e., that the objective function value over the model trained using our selective strategy is at least as good as the objective value obtained with full training over all data samples over all training cycles.

This can be seen as a balance between two forces: (1.) high MFO cutoff means filtering only data where we are sure of the label, and do not mistakenly dismiss information and in turn decrease model accuracy. (2.) low MFO cutoff means reducing training to fewer instances, thus fewer minibatches and finally lower training time. Clearly, the best filtering cutoff is a trade-off. We propose to study the decrease in MFO in the log-scale, and define $\Delta MFO = -\log_{10}(1 - MFO)$. Our empirical study suggests values in the range $2 \geq \Delta MFO \geq 1.5$. At prediction time we match new data to clusters. For data in removed clusters we use its dominating class label. For other clusters, we use the model trained on the reduced set. Algorithm 1 outlines our selective training

strategy. In the experiments, we demonstrate that our approach indeed converges faster, i.e., uses fewer training epochs and converges to an accuracy that is at least as high as the one for full time training on all data.

Algorithm 1 Proposed fast training approach

procedure
  $D \leftarrow$ corpus of labeled documents
  $k \leftarrow$ Number of clusters to generate
  $MFO_{cut} \leftarrow$ cutoff for filtering
  while *pretraining* **do**
    $\Delta Acc \leftarrow$ rate of acc. improv.
    if ΔAcc starts dropping **then**
      break pretraining
  Cluster using K-means
  $MFO_i \leftarrow MFO$ for cluster i
  $D' \leftarrow$ clusters with $MFO_i \leq MFO_{cut}$
  while *main-training* **do**
    train on reduced dataset D'
    if convergence is achieved **then**
      break main-training

5 Evaluation

5.1 Data and experimental setup

We use the large-scale, open access Enron data [18], comprised of more than $500,000$ documents that vary greatly in style, language and length and thus provide excellent insight into performance on varied text. All documents are split into sentences using the Punkt sentence boundary detection approach [16]. Constituency parse trees (splitting sentences into phrases) are generated from a probabilistic context-free grammar [17], trained over the Stanford Penn Treebank [32]. We train a recursive neural network model over these parse trees, which allows us to learn an embedding for each phrase. We make this data available online[6].

We use labels by domain experts from the TREC competition [6, 33], where topics were labeled by at least 3 human annotators (using majority where different). We evaluate on 4 binary topics where a sentence is *true* if it belongs to the topic and *false* otherwise: $FCAST$**:** 267366 sentences regarding Enron's financial state. We use 40000 sentences for validation, 40000 for testing, the rest for training. The percentage of positive (i.e., true) sentences is 31%. FAS**:** 178266 sentences where Enron claims compliance with Financial Accounting Standards[7]. We use 27000 sentences for validation, 27000 for training; 59% positive sentences. $PPAY$**:** 134256 sentences about financial *prepay transactions*. We use 15000 sentences for validation, 15000 for testing; 13% positive

[6] code `https://bitbucket.alexandra.dk/projects/TAB`, data `https://dataverse.harvard.edu/dataverse/enron-w-trees`

[7] `http://www.fasb.org/jsp/FASB/Document_C/DocumentPage?cid=1218220124871`

sentences. $EDENCE$**:** 167913 sentences discussing tampering with evidence. We use 25000 sentences for validation, 25000 for testing; positive sentences 23%. For further details see [6, 33, 24].

We test Selective Training on an RecNN where the intrinsic model is applied recursively over parse-trees. For long sentences the number of layers in the final recursive network ("unfolded" in time and structure) reaches several hundreds layers. We report findings where the intrinsic model has a single hidden layer and where the hidden layer has 100 neurons. We have tested on a smaller development set and found these hyperparameters to yield robust performance. Standard full training takes in the order of 1 day per $1,000,000$ minibatches, whereas clustering takes few minutes. Minibatch count is thus an appropriate measure for training time of full and selective training .

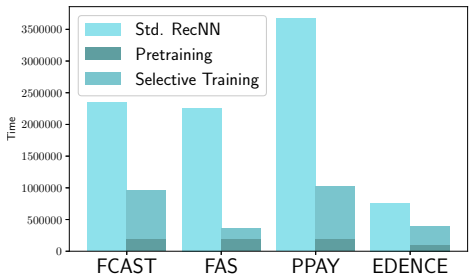

Fig. 1. Training time (in mini-batches) on 4 datasets

5.2 Empirical study and discussion

Fig. 1 reports training times on 4 different datasets of varying complexities. Standard backpropagation through structure (denoted Std. RecNN in the figure) converges on the $EDENCE$ dataset in $758,000$ minibatches, and on the $PPAY$ dataset in $3,674,000$ minibatches, even though the datasets are of comparable size. Our approach reduces the training time by a factor of approximately 2 to 6, depending on the dataset under study. $FCAST$ and FAS have approximately the same runtime on the full dataset, which our approach reduces for $FCAST$ by factor 2.42, while for FAS we obtain an impressive factor of 6.1. It seems that FAS can be learned from fewer training instances, which our method picks up on.

It is interesting to note that our approach indeed selects the most relevant sets of samples for training, even being able to slightly improve accuracy. Comparing standard full training and our selective training strategy (in parentheses), we have $FCAST$ 83.24% (83.41%), FAS 96.05% (96.40%), $PPAY$ 95.93% (96.09%) and $EDENCE$ 89.02% (89.12%).

Stopping pre-training, i.e., when to cluster and remove instances, can be determined from the graph over accuracy as a function of training time (see Fig. 2 for accuracy on $PPAY$ using backpropagation; other datasets show similar behaviour; omitted here due to space limitations).

Vertical cuts on the training graph show pretraining stopping points and the number of additional mini-batch visits required after filtering. The minimum (*Total* column)

Pretraining	Training	Total
0	3,674,000	3,674,000
50,000	2,128,000	2,178,000
100,000	2,365,000	2,465,000
150,000	913,000	1,063,000
200,000	**830,000**	**1,030,000**
250,000	1,036,000	1,286,000
500,000	1,818,000	2,318,000

Table 1. Pretraining cutoff on $PPAY$

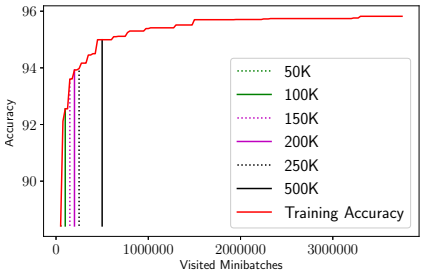

Fig. 2. Pre-train cutoffs, $PPAY$ training curve

is at 200,000 mini-batches. Comparing with cut-lines in Figure 2, we note that the best stopping point for pretraining is where the curve "bends", i.e., where the rate of improvement starts to plateau. At this point we have gained most (further gains are more expensive) and thus have the best potential for out-performing full training. At this point, large-scale statistical properties of the data are encoded.

To generate the curve in Figure 2, we use the "pocket" algorithm where the best performing model seen so far is used to output the accuracy on a small validation set. This process yields a nice monotonically increasing curve and allows for a simple pre-training cutoff criteria. In our experiments, we stopped training manually based on the curve, but this could be automatized based on the shape and slope of the curve.

MFO	ΔMFO	Count	Percentage	Total Training
-	(0)	-	-	3,674,000
0.9970	2.52	1	3.6%	3,362,000
0.9945	2.26	3	11.9%	2,574,000
0.9940	2.22	4	19.3%	1,194,000
0.9867	1.88	5	23.0%	1,030,000
0.9863	1.85	6	26.9%	891,000
0.9858	1.85	8	34.3%	1,480,000
0.9800	1.70	12	50.6%	1,739,000
0.9720	1.55	15	59.9%	>5,000,000

Table 2. Filter percentage, $PPAY$, $200K$ minibatches

.

To determine how many clusters should be filtered out, we study the model after 200,000 minibatches and its sentence representations. These representations are clustered using K-means and we filter out all clusters with a higher MFO ratio than a filter cutoff. Figure 3 shows clusters sorted by the ratio of sentences with class 1 (using the simplified approach for 2-class problems as described above). Clusters to be filtered (i.e., high MFO ratios) are at the far left and right and clusters to keep are at the center. Note that the MFO ratio for the far left clusters is much higher than the MFO ratio for the far right clusters. For dataset $PPAY$ it seems easier finding pure clusters of label 0, whereas clusters with a high ratio of label 1 tend to be mixed with many examples of label 0 occurrences. Thus, here we only filter clusters on the far left.

We show converge times for different filtering cutoffs from 1 cluster (3.6% of data filtered) to 15 clusters (59.9%) in Table 2. We observe marked improvements in the range 19.3% to 26.9%. Cutoffs are shown as vertical lines against class 1 ratio in Fig. 3.

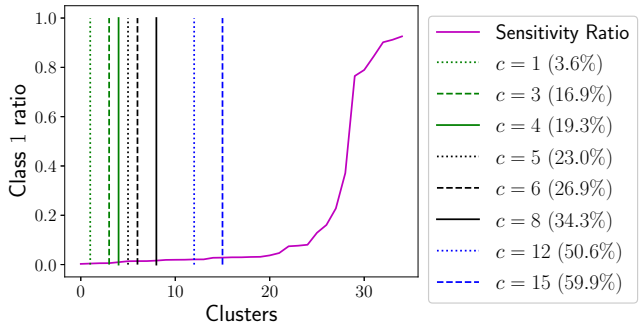

Fig. 3. Filter cutoffs (Table 2) over class 1 ratio.

Table 2 shows a large drop in ΔMFO from 2.22 to 1.88, where we thus should place our cut to obtain a runtime of $1,194,000$, which is significantly less than the standard full training runtime of $3,674,000$.

Table 3 shows robustness to hyperparameter k. We test values between 15 and 70 and compare runtime of 3 different filtering cuts. Each cut is a row in Table 3 ("Small (S)", "Medium (M)", "Large (L)"). "Medium (M)" corresponds to the optimal size cut and the others are smaller and larger, respectively. We set the MFO cutoff such that we filter approximately the same amount of examples across different k values (ratio of filtered samples given in column "Size"). Please note that the actual ratio of samples filtered varies only slightly, due to filtering an integral number of clusters, but still allowing a comparison of runtimes across different values of k.

Across k values we experiment with filtering a "Small (S)" amount ($\approx$ 12%) of data, a "Medium (M)" amount ($\approx$ 27%) and a "Large (L)" amount ($\approx$ 33%). Table 3 shows that runtimes only vary slightly, i.e., an average difference from mean of 6.32%, 6.19%, 11.21% for Small, Medium and Large, respectively. Thus, k does not impact reduction in training time significantly.

	$k = 15$		$k = 35$		$k = 70$		
	Size	Time	Size	Time	Size	Time	Avg. Diff
S	11.92%	$1,486K$	11.91%	$1,683K$	11.87%	$1,756K$	6.32%
M	27.84%	$924K$	26.92%	$891K$	26.92%	$1,040K$	6.19%
L	33.66%	$1,551K$	34.32%	$1,480K$	33.07%	$1,933K$	11.21%

Table 3. Time (measured as number of minibatches) over k and filtering cutoffs (Small (S), Medium (M) and Large (L)). Size is filtered data ratio

6 Detailed Error Analysis

We study whether selective training still learns models that generalize as well as full training models do. We report prediction accuracy in $PPAY$ (76067 sentences with

embeddings from both selected and full training set) in the interest of space; findings are similar for the other datasets.

Table 4 shows that the standard full training model misclassifies 1926 instances, whereas our approach only misclassifies 1565. Out of these 1565, 869 are made by the full training approach as well, i.e., our approach makes 18.74% fewer errors and out of the remaining, 55% are identical to those of full training. We conclude that our speed up does not jeopardize accuracy, but even leads to slightly improved accuracy. We analyze errors using K-means clustering of embeddings (Table 5). Here *Id* refers

Total number of shared embeddings	76067
Errors of the standard training	1926
Accuracy of the standard training	97.47%
Errors of our training method	1565
Accuracy of our training method	97.94%

Table 4. Shared embeddings in $PPAY$.

Id	Size	Accuracy
1	97	100.0%
2	113	100.0%
3	112	99.1071%
4	188	98.9362%
5	134	89.5522%
6	202	78.7229%
7	310	74.8387%
8	78	71.0526%
9	185	48.6486%
10	148	41.8919%

Table 5. Error statistics. See text for details.

to cluster id. *Size* is the number of elements (errors) in a cluster and *Accuracy* the overall accuracy when we cluster all data elements (not just errors) into these 10 clusters and calculate the accuracy of our approach per cluster. The errors are fairly uniformly distributed; we observe that except for group 7 with 310 elements, all other clusters have size around $100 - 200$. This uniform size distribution (especially compared to Fig. 3) suggests that the errors are not concentrated in any particular part of the embedding space. Our approach thus is expected to generalize well.

A concern for the clustering step could be that K-Means fails to do meaningful clustering for very high representation sizes. Such investigation falls outside the work done here. However we observe that this degradation of K-means has been investigated in [2] where the authors find that hierarchical K-means may provide strong clustering even for high dimensionalities.

Before studying actual examples, we note that the first four groups in Table 5 contain 510 out of 1565 of our errors (32.59%) with high MFO, all for label 1, which makes some of the errors in these groups challenging to resolve. Groups $6, 7, 8$ have a medium MFO score, with group 5 showing a slightly higher score. Finally, groups $\{9, 10\}$ show lower MFO. From Table 6 with error examples from each group, we observe the following types of errors: **Soft errors.** Clusters $2, 3, 4$ contain examples of prepay transactions (label 1) where wording and structure seem to be good indicators of the class. These clusters also contain challenging samples. Cluster 7 seems similar, but shows greater diversity in label distribution, which is even more challenging to resolve. **Poor filtering.** Cluster 8 contains examples of sentences which should have been filtered out in pre-processing, and can therefore be used to inform the pre-processing step. **Short emails and headers.** Clusters $6, 9$ contain examples of short email sentences.

Again, this can be used to inform pre-processing e.g. combining them with email subject, to/from or the like. **Hard errors.** Clusters $1, 5, 10$ contain headlines, locations and sentences with little information, which we consider unlikely targets for improvement based on sentences alone.

Group Id	Sentence	Type of error
1	Section 12.1 Duration	Document headlines
2	I spoke to Tim today regarding the prepayment transactions with terminated counterparties	Generic sentence
3	Let me know, but as I understand it, the Chase agreement should be a good format for this master agreement	Stating facts
4	A similar swap was entered into between the American Public Energy Agency and Chase on the same date	Prepay technicalities
5	Maybe this happens in a region of not prepay and that is why... New York, New York 10043	Specific location
6	If there is anything else you need please feel free to call me on 0207 783 5404	Short standard email
7	Please let me know the nature of the transaction with National Steel	General question wrt. entity
8	[. . .] – – – – – - Enron-6.11.00.ppt	Bad text/filtering
9	RE: Swap Transaction; Deal No M180816	Preambles, email header
10	The PSCO Project will be sold into TurboPark on 1/19/01	Abbreviation and numbers

Table 6. Samples from each of the 10 error groups with description of type of error

In summary, some groups can be used directly to improve performance further in pre-processing, others offer potential for future work, and few contain too little information to be correctly predicted. Models trained using our faster selective training strategy seem to generalize just as well as full training, providing even slightly better accuracy.

7 Related Work

There is a large body of research on different training strategies, often with the aim of improving the accuracy or stability of a classifier. The classical approach is AdaBoost [8] which adapts training depending on the error observed by weighing "difficult" examples higher. At an abstract level, boosting also refocuses training. However, there are two core differences between our approach and boosting: first, we aim to reduce training time (i.e., speeding up convergence of the model parameters) by removing samples that are not expected to benefit. And secondly, we base the selection of samples on their multi-dimensional representation rather than on the one-dimensional difference in ground truth label and predicted label. For an extensive survey of bagging and boosting in classification we refer the interested reader to [19].

Adaptive learning methods, such as Adam [14], automatically adjust the learning rate based on adaptive estimates of lower-order moments in the loss function. [14] observe faster convergence of the model during training to a particular accuracy (cost) value compared to other popular learning optimizers (AdaGrad, RMSProp, SGDNesterov, AdaDelta). Adaptive learning methods thus essentially adjust the learning rate,

whereas our approach takes a fundamentally different approach that adapts the training data being used to learn accurate models much more efficiently. [22] estimates which samples provide largest decrease in loss function based on estimates of previous decreases incurring an $N \log N$ sorting/rank penalty over all samples. By contrast, we use the rich information embedded in the representations, using clustering for easy selection. Zhao et al. [34] use K-means clustering to differentiate local from global word context for improved text embeddings, but not for training time reduction.

8 Conclusion and Future Work

We present an efficient selective strategy for reducing training time of complex models, focusing on recurrent neural networks over phrase trees on large text datasets. We propose studying groups in representation space to identify where learning from training data seems to be completed, and where more training is expected to improve model accuracy. Discarding clusters with pure label distribution, we refocus training to those samples that lead to high accuracy models with less training time. We show how to easily infer the parameters for selecting clusters using rate of improvement on training graphs and our proposed ΔMFO measure. In thorough experiments, we demonstrate up to 6 times faster training without loss of accuracy on a number of datasets.

Our method generalizes to similar training problems of complex models that generate distributed instance representations. We intend to study representation based models, such as Recurrent Neural Networks, Long Short Term Memory Networks and Convolutional Neural Networks.

Acknowledgments This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 732240 (Synchronicity Project). The authors would like to thank the anonymous reviewers for valuable comments and suggestions.

References

1. Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. JMLR **3**, 1137–1155 (2003)
2. Coates, A., Ng, A.Y.: Learning feature representations with k-means. In: Montavon, G., Orr, G., Müller, K. (eds.) Neural Networks: Tricks of the Trade, LNCS, vol. 7700, pp. 561–580. Springer (2012)
3. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ICML. pp. 160–167 (2008)
4. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. JMLR **12**, 2493–2537 (2011)
5. Conneau, A., Kiela, D., Schwenk, H., Barrault, L., Bordes, A.: Supervised learning of universal sentence representations from natural language inference data. In: EMNLP. pp. 670–680 (2017)
6. Cormack, G.V., Grossman, M.R., Hedin, B., Oard, D.W.: Overview of the trec 2010 legal track. In: TREC (2010)

7. Forgy, E.W.: Cluster analysis of multivariate data: Efficiency versus interpretability of classification. Biometrics **21**(3), 768–769 (1965)
8. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: ICML. pp. 148–156 (1996)
9. Goller, C., Kuchler, A.: Learning task-dependent distributed representations by backpropagation through structure. In: IEEE ICNN. pp. 347–352 (1996)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8) (1997)
12. İrsoy, O., Cardie, C.: Deep recursive neural networks for compositionality in language. In: NIPS. pp. 2096–2104 (2014)
13. Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP. pp. 1746–1751 (2014)
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015)
15. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017)
16. Kiss, T., Strunk, J.: Unsupervised multilingual sentence boundary detection. Comp. Ling. **32**(4), 485–525 (2006)
17. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: ACL. pp. 423–430 (2003)
18. Klimt, B., Yang, Y.: The Enron corpus: A new dataset for email classification research. In: ECML. pp. 217–226 (2004)
19. Kotsiantis, S.B.: Bagging and boosting variants for handling classifications problems: a survey. Knowl. Eng. Rev. **29**(1), 78–100 (2014)
20. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: ICML. pp. 1188–1196 (2014)
21. Lloyd, S.: Least squares quantization in PCM. IEEE TIT **28**(2), 129–137 (1982)
22. Loshchilov, I., Hutter, F.: Online batch selection for faster training of neural networks. In: ICLR Workshop (2016)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS. pp. 3111–3119 (2013)
24. Neerbek, J., Assent, I., Dolog, P.: Detecting complex sensitive information via phrase structure in recursive neural networks. In: PAKDD (2018)
25. Olvera-López, J.A., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F., Kittler, J.: A review of instance selection methods. AI Rev. **34**(2), 133–143 (2010)
26. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. pp. 1532–1543 (2014)
27. Rush, A.M., Chopra, S., Weston, J.: A neural attention model for abstractive sentence summarization. In: EMNLP (2015)
28. Settles, B.: Active learning. Synthesis Lectures on Artificial Intelligence and Machine Learning **6**(1), 1–114 (2012)
29. Socher, R., Huang, E.H., Pennin, J., Manning, C.D., Ng, A.Y.: Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In: NIPS. pp. 801–809 (2011)
30. Socher, R., Manning, C.D., Ng, A.Y.: Learning continuous phrase representations and syntactic parsing with recursive neural networks. In: NIPS Deep Learn. Unsup. Feat. Learn. WS. pp. 1–9 (2010)
31. Socher, R., Perelygin, A., Wu, J.Y., Chuang, J., Manning, C.D., Ng, A.Y., Potts, C.: Recursive deep models for semantic compositionality over a sentiment treebank. In: EMNLP. pp. 1631–1642 (2013)
32. Taylor, A., Marcus, M., Santorini, B.: The penn treebank: An overview. In: Abeillé, A. (ed.) Treebanks, pp. 5–22. Text, Speech and Language Technology, Springer (2003)
33. Tomlinson, S.: Learning task experiments in the trec 2010 legal track. In: TREC (2010)
34. Zhao, Z., Liu, T., Li, B., Du, X.: Cluster-driven model for improved word and text embedding. In: ECAI. pp. 99–106 (2016)